École Nationale Supérieure de Techniques Avancées - ENSTA Paris

Optimisation du chemin minimal entre aérodromes

Christopher Tsa-Kwet-Shin, Edouard Raimbault

Palaiseau

Octobre 2024

Table des matières

1 Introduction

Le problème étudié est une variante du problème du voyageur de commerce (TSP), adaptée au contexte de planification de trajets aériens. Le but est de trouver le chemin le plus court entre deux aérodromes (départ et arrivée) dans un ensemble de régions contenant des aérodromes en respectant des contraintes spécifiques :

- Une distance maximale entre les trajets est imposée (contrainte de distance maximale).
- L'avion doit passer par un nombre minimal A_{min} d'aérodromes (contraintes du nombre minimum d'aérodromes visités).
- Si l'avion atteint un aérodrome autre que celui d'arrivée il doit repartir vers un autre aérodrome (contrainte de conservation du flux).
- Il n'est pas nécessaire de visiter tous les aérodromes, mais toutes les régions doivent être couvertes (contrainte de couverture régionale).
- Le chemin n'est pas un cycle, il commence à l'aérodrome de départ et se termine à l'aérodrome d'arrivée (contrainte de départ et d'arrivée).

Ce rapport décrit les différentes formulations mathématiques du problème et présente les techniques de résolution implémentées en Julia avec JuMP et Gurobi.

2 Formulation mathématique

2.1 Variables de décision et définition des ensembles

- V : ensemble des aérodromes présent sur la carte de cardinal n (dans notre cas il s'agit de l'ensemble $\{1, 2, \ldots, n\}$).
- A : ensemble des trajets entre 2 aérodromes appartenant à V de cardinal m (dans notre cas, il s'agit de l'ensemble $\{(i, j) \mid (i, j) \in \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\} \text{ et } i \neq j\}$).
- $x_{ij} \in \{0, 1\}, \forall (i, j) \in A$: variable binaire valant 1 si l'arc entre les aérodromes i et j appartenant à V fait parti du trajet.
- $u_i \geq 0, \forall i \in \{1, 2, \ldots, n\}$: variable continue introduite dans la formulation MTZ pour éliminer les sous-tours.

2.2 Objectif

L'objectif du problème est de minimiser la distance totale parcourue :

$$\min \sum_{(i,j) \in A} D_{ij} \cdot x_{ij}$$

où D_{ij} est la distance entre les aérodromes i et j.

2.3 Contraintes

On note $\delta^+(i)$ et $\delta^-(i)$ les ensembles des arcs partant et sortant du nœud i, par $\delta^+(S)$ et $\delta^-(S)$ les arcs sortant/entrant de $S \subseteq V$, et par $A(S)$ les ensembles des arcs avec les deux extrémités dans $S \subseteq V$. Définissons aussi $V_i := V \setminus \{i\}$, $V_{ij} := V \setminus \{i, j\}$. Notons d l'indice de l'aérodrome de départ et f l'indice de celui d'arrivée.

2.3.1 Constraintes générales

Nous pouvons mettre en équation les contraintes du problème de la manière suivante :

— **Distance maximale** : Chaque trajet entre deux aérodromes doit respecter une distance maximale R :

$$D_{ij} \cdot x_{ij} \leq R \quad \forall (i,j) \in A$$

— **Nombre minimum d'aérodromes visités** :

$$\sum_{(i,j)\in A} x_{ij} \geq A_{\min} - 1$$

— **Couverture régionale** : Au moins un aérodrome dans chaque région doit être visité :

$$\sum_{\substack{(i,j)\in A \\ j\in \text{regions}[k]}} x_{ij} \geq 1 \quad \forall k \in \{1, 2, \ldots, Nr\}$$

où Nr est le nombre de régions et $\forall k \in \{1, 2, \ldots, Nr\}$, regions$[k]$ est l'ensemble des points de V appartenant à la région k.

— **Conservation du flot** :

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ji}, \quad \forall i \in \{1, \ldots, n\} \text{ avec } i \neq d, f$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} \leq 1, \quad \forall i \in \{1, \ldots, n\}$$

— **Départ et arrivée** : Le chemin commence à d et se termine à f :

$$\sum_{\substack{j=1 \\ j\neq d}}^{n} x_{dj} - \sum_{\substack{j=1 \\ j\neq d}}^{n} x_{jd} = 1$$

$$\sum_{\substack{j=1 \\ j\neq f}}^{n} x_{fj} - \sum_{\substack{j=1 \\ j\neq f}}^{n} x_{jf} = -1$$

2.3.2 Contrainte d'élimination des sous-tours avec la formulation MTZ (Miller-Tucker-Zemlin)

Dans le cas d'une formulation avec un nombre polynomial de contraintes, nous pouvons écrire les contraintes d'élimination des sous-tours de la manière suivante :

$$u_i - u_j + n \cdot x_{ij} \leq n - 1 \quad \forall (i,j) \in A \text{ tel que } i, j \neq d$$

$$\text{et } u_d = 0$$

2.3.3 Contrainte d'élimination des sous-tours avec la formulation DFJ (Dantzig–Fulkerson–Johnson)

Dans le cas d'une formulation avec un nombre exponentiel de contraintes, nous écrivons la contrainte d'élimination des sous-tours de la manière suivante :

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subset \{1, \ldots, n\}, \quad 2 \leq |S| \leq n - 1$$

2.3.4 Formulations alternatives issues de l'article (Taccari, 2016)

Nous avons également les quatre formulations alternatives suivantes :

— **GCS (Generalized Cutset Inequalities)** : cette formulation utilise la contrainte suivante pour éliminer les sous-tours :

$$\sum_{(i,j) \in \delta^+(S)} x_{ij} \geq \sum_{(k,j) \in \delta^+(k)} x_{kj} \quad \forall k \in S, \forall S \subseteq V_{df}, \quad |S| \geq 2$$

— **RLT (Reformulation-linearization based formulation)** :

$$\alpha_{ij} = \beta_{ij} + x_{ij} \quad \forall (i,j) \in A \tag{1}$$

$$x_{dj} + \sum_{(i,j) \in \delta^-(j), i \neq d} \alpha_{ij} - \sum_{(j,i) \in \delta^+(j)} \beta_{ji} = 0 \quad \forall j \in V_t, (d,j) \in \delta^+(d) \tag{2}$$

$$\sum_{(i,j) \in \delta^-(j)} \alpha_{ij} - \sum_{(j,i) \in \delta^+(j)} \beta_{ji} = 0 \quad \forall j \in V_{st}, (d,j) \notin \delta^+(d) \tag{3}$$

$$\alpha_{ij} \leq (n-1) x_{ij} \quad \forall (i,j) \in A, i \neq d \tag{4}$$

$$x_{ij} \leq \beta_{ij} \quad \forall (i,j) \in A, i \neq d \tag{5}$$

$$\alpha_{ij} \geq 0, \quad \beta_{ij} \geq 0 \quad \forall (i,j) \in A \tag{6}$$

— **SF (Single-flow Formulation)** : cette formulation utilise une contrainte de capacité et 3 contraintes d'équilibre sur les flows pour éliminer les sous-tours :

$$q_{ij} \leq (n-1)x_{ij} \quad \forall (i,j) \in A \tag{1}$$

$$\sum_{(s,j)\in\delta^+(s)} q_{sj} = \sum_{k\in V_s} z_k \tag{2}$$

$$\sum_{(i,k)\in\delta^-(k)} q_{ik} - \sum_{(k,j)\in\delta^+(k)} q_{kj} = z_k \quad \forall k \in V_s \tag{3}$$

$$\sum_{(i,k)\in\delta-(k)} x_{ik} = z_k \quad \forall k \in V_s \tag{4}$$

$$q_{ij} \geq 0 \quad \forall (i,j) \in A \tag{5}$$

$$z_k \in \{0,1\} \quad \forall k \in V_s \tag{6}$$

— **MCF (Multicommodity-flow formulation)** : cette formulation est une variante de la SF qui remplacent les flows q définis dans la SF par n-1 flows unitaires.

$$q_{ij}^k \leq x_{ij}, \quad \forall k \in V_d, \quad (i,j) \in A \tag{1}$$

$$\sum_{(i,j)\in\delta^+(i)} q_{ij}^k - \sum_{(j,i)\in\delta^-(i)} q_{ji}^k = \begin{cases} z_k, & \text{si } i = d \\ -z_k, & \text{si } i = k \\ 0, & \text{sinon} \end{cases} \quad \forall i \in V, \quad \forall k \in V_d \tag{2}$$

$$\sum_{(i,k)\in\delta^-(k)} x_{ik} = z_k, \quad \forall k \in V_d \tag{3}$$

$$\sum_{(d,j)\in\delta^+(d)} x_{dj} = 1 \tag{4}$$

$$\sum_{(i,f)\in\delta^-(f)} x_{if} = 1 \tag{5}$$

$$q_{ij}^k \geq 0, \quad \forall k \in V_d, \quad (i,j) \in A \tag{6}$$

$$z_k \in \{0,1\}, \quad \forall k \in V_d \tag{7}$$

2.4 Récapitulatif des nombres de variables et de contraintes des formulations utilisées

Formulation	Nombre de variables	Nombres de contraintes
MTZ	$O(m)$	$O(m)$
DFJ	$O(m)$	$O(2^n)$
GCS	$O(m)$	$O(n^2 2^n)$
RLT	$O(m)$	$O(m)$
SF	$O(m)$	$O(m)$
MCF	$O(m)$	$O(nm)$

TABLE 1 – Tableau récapitulatif des nombres de variables et de contraintes

3 Techniques de résolution en Julia

Le résolution a été implémentée en Julia avec JuMP et Gurobi comme solveur. Afin de tester nos différentes méthodes de résolution, nous avons à disposition 11 instances allant de 6 à 100 aérodromes.
La principale difficulté du problème réside dans l'élminination des sous-tours d'une solution de vol. En effet, sans contrainte d'élimination des sous-tours, la résolution de l'instance 20_1 donne la solution suivante :

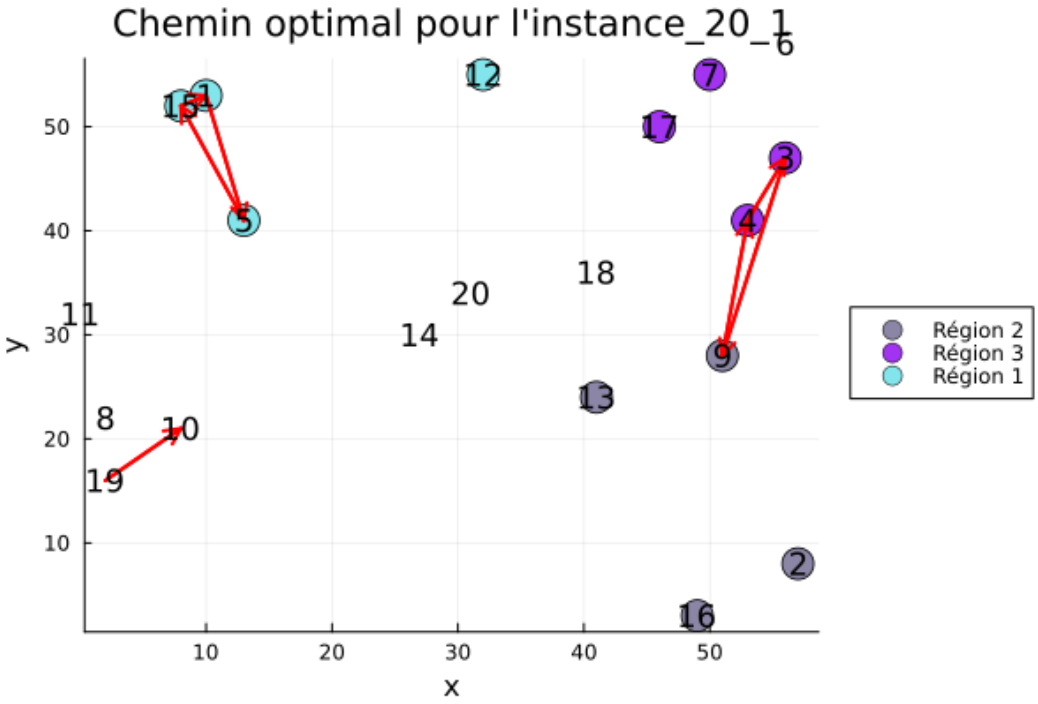

FIGURE 1 – Solution sans contrainte d'élimination pour l'instance 20_1

Ainsi, différentes méthodes d'élimination des sous-tours ont été expérimentées dans les sous-sections suivantes à l'aide des formulations de la section 2. Pour chaque formulation, le corps général du code se compose d'une fonction *resolution* qui résout le programme linéaire en nombres binaires et d'une fonction *visualize_solution* qui trace sur une carte 2D le parcours de l'avion correspondant à la solution issue de *resolution*. Les tracées des parcours optimaux des 11 instances sont en annexes.

3.1 Résolution avec la formulation MTZ

Dans la cadre de la formulation MTZ, nous avons implémenté la contrainte d'élimination des sous-tours en utilisant la variable continue u de la manière suivante :

```
# Variables de MTZ pour eliminer les sous-tours
@variable(model, u[1:n] >= 0)

# Contraintes de la formulation MTZ
@constraint(model,
[i in 1:n, j in 1:n; i != j && i != d && j != d], u[j] >= u[i] + 1 - n*(1 - x[i,j]))
@constraint(model, u[d] == 0)
```

Listing 1 – Contraintes d'élimination des sous-tours de la formulation MTZ

3.2 Résolution avec la formulation DFJ

La résolution du problème avec un nombre exponentiel de contraintes implique d'implémenter une méthode de détection des sous-ensembles d'aérodromes formant des sous-tours. Ainsi, une méthode naïve et une méthode avec un problème de séparation ont été testées. Quelque soit la méthode de détection, la fonction *add_initial_constraints* permet d'éliminer les sous-tours sur des sous-ensembles de 2 et 3 noeuds. Pour chaque méthode, on commence par résoudre le problème sans contrainte d'élimination des sous-tours. Puis on applique la méthode de détection et on résout le problème augmentés des contraintes d'élimination correspondant aux sous-tours détectés. On répète ces opérations d'élimination de sous-tours jusqu'à résolution.

3.2.1 Détection des sous-tours par méthode naïve

La méthode naïve consiste à résoudre le problème sans contrainte d'élimination des sous-tours pour ensuite détecter les sous-tours présents dans la solution et rajouter une contrainte de l'élimination pour chaque sous-tour identifié. Nous avons ainsi implémenter une fonction de détection des sous-tours prenant en entrée la solution du problème sans contrainte d'élimination des sous-tours et réalisant les étapes suivantes :

— Construction d'un dictionnaire des arcs traversés par la solution

— Pour chaque aérodrome de l'instance, on prolonge tant que les arcs sont traversés par la solution et on détecte le sous-tour si un arc traversé revient vers cet aérodrome

— Pour chaque sous-tour détecté, on ajoute une contrainte d'élimination sur le sous-ensemble d'aérodromes associé à ce sous-tour.

```
# Contraintes de la formulation DFJ
@constraint(model, sum(x[i, j] for i in subtour, j in subtour if i != j) <= length(subtour)
    - 1)
```

Listing 2 – Contraintes d'élimination des sous-tours pour la formulation DFJ

3.2.2 Détection des sous-tours par problème de séparation

La deuxième méthode de détection de sous-tours consiste à résoudre le problème suivant :

$$\max(\sum_{(i,j)\in V} x_{ij}^{sol} \cdot z_i \cdot z_j) - (\sum_{i\in V} z_i - 1)$$

où

— x^{sol} correspond à la solution actuelle du problème.

— les $z_i, \forall i \in V$ sont des variables binaires qui valent 1 si i est dans S et 0 si i n'est pas dans S.

Ainsi, pour chaque solution du problème de l'avion, on résout le problème de séparation ci-dessus. Si la valeur objective de ce problème est possitive alors la solution contient un sous-tour qui est composé de tous les noeuds i pour lesquels $z_i = 1$. Puis en enlevant les sommets correspondants au sous-ensemble du sous-tour et on continue à itérer tant que d'autres sous-tours sont détectés.

3.3 Résolution avec la formulation RLT

Pour la formulation RLT, nous avons repris le code de la formulation MTZ où nous avons ajouté dans la fonction *resolution* les contraintes suivantes :

```
# Variables de RLT pour eliminer les sous-tours
@variable(model, a[i=1:n, j=1:n; i != j] >= 0)  # a_{ij} = u_j * x_{ij}
@variable(model, b[i=1:n, j=1:n; i != j] >= 0)  # b_{ij} = u_i * x_{ij}
@variable(model, u[i=1:n] >= 0)

# Contraintes de la formulation RLT
@constraint(model, [i in 1:n, j in 1:n; i != j], a[i, j] == b[i, j] + x[i, j])
@constraint(model, [j in 1:n; j != d && j != f], x[d, j] +
sum(a[i, j] for i in 1:n if i != j && i != d) - sum(b[j, i] for i in 1:n if i != j) == 0)
@constraint(model, [i in 1:n, j in 1:n; i != j && i != d], a[i, j] <= (n - 1) * x[i, j])
@constraint(model, [i in 1:n, j in 1:n; i != j && i != d], x[i, j] <= b[i, j])
@constraint(model, u[d] == 0)
@constraint(model, [i in 1:n, j in 1:n; i != j], u[i] + 1 <= u[j] + n * (1 - x[i,j]))
@constraint(model, [i in 1:n, j in 1:n; i != j], u[j] + 1 <= u[i] + n * (1 - x[j,i]))
```

Listing 3 – Contraintes d'élimination des sous-tours pour la formulation RLT

3.4 Résolution avec la formulation GCS

Le code pour la formulation GCS est similaire à celui de la formulation DFJ avec détection par méthode de séparation à la différence que la contrainte ajouté au problème lors de la détection d'un sous-tour est la suivante :

```
# Ajout des contraintes GCS pour interdire le sous-tour
@constraint(model, [k in subtour], sum(x[i, j] for i in subtour, j in subtour if i != j)
<= sum(x[i, j] for i in subtour if i != k for j in 1:n))
```

Listing 4 – Contraintes d'élimination des sous-tours pour la formulation GCS

3.5 Résolution avec la formulation SF

La code correspondant à la formulation SF contient comme contrainte d'élimination les contraintes de flows suivantes :

```
# Variables de SF
@variable(model, 0 <= q[i=1:n, j=1:n])  # Auxiliary flow variable for SF
@variable(model, z[i=1:n; i!=d], Bin)  # Binary variable for nodes in path

# Contraintes de la formulation SF
@constraint(model, [i in 1:n, j in 1:n; i != j], q[i, j] <= (n - 1) * x[i, j])
@constraint(model, sum(q[d, j] for j in 1:n if j != d) == sum(z))
@constraint(model, [k in 1:n; k != d], sum(q[i, k] for i in 1:n if i != k)
- sum(q[k, j] for j in 1:n if j != k) == z[k])
@constraint(model, [k in 1:n; k != d], sum(x[i, k] for i in 1:n if i != k) == z[k])
```

Listing 5 – Contraintes d'élimination des sous-tours pour la formulation SF

3.6 Résolution avec la formulation MCF

La code correspondant à la formulation MCF contient comme contrainte d'élimination les contraintes de flows suivantes :

```
# Variables de MCF
@variable(model, 0 <= q[k=1:n, i=1:n, j=1:n; k!=d])  # Auxiliary flow variable for SF
@variable(model, z[i=1:n; i!=d], Bin) # Binary variable for nodes in path

# Contraintes de la formulation SF
@constraint(model, [k in 1:n, i in 1:n, j in 1:n; k != d && i != j], q[k, i, j] <= x[i, j])
@constraint(model, [k in 1:n, i in 1:n; k != d && i != d && i != k],
sum(q[k, i, j] for j in 1:n if j != i) - sum(q[k, j, i] for j in 1:n if j != i) == 0)
@constraint(model, [k in 1:n; k != d], sum(q[k, d, j] for j in 1:n if j != d)
- sum(q[k, j, d] for j in 1:n if j != d) == z[k])
@constraint(model, [k in 1:n; k != d], sum(q[k, k, j] for j in 1:n if j != k)
- sum(q[k, j, k] for j in 1:n if j != k) == - z[k])
@constraint(model, [k in 1:n; k != d], sum(x[i, k] for i in 1:n if i != k) == z[k])
```

Listing 6 – Contraintes d'élimination des sous-tours pour la formulation MCF

3.7 Présentation des performances

Instances	Instance_6_1	Instance_20_1	Instance_20_2	Instance_20_3	Instance_30_1	Instance_40_1	Instance_50_1	Instance_70_1	Instance_80_1	Instance_80_2	Instance_100_1
Valeur optimale	12	122	102	106	139	112	163	436	488	527	438

FIGURE 2 – Valeur objective optimale par instance

Méthodes de résolution	Instance_6_1	Instance_20_1	Instance_20_2	Instance_20_3	Instance_30_1	Instance_40_1	Instance_50_1	Instance_70_1	Instance_80_1	Instance_80_2	Instance_100_1
Formulation MTZ	8	36.3	36.1	35.1	53.4	67.35	66.44	Non résolu	Non résolu	Non résolu	8
Formulation DFJ avec détection naïve	8	Non résolu	59	Non résolu	Non résolu	81	98	Non résolu	Non résolu	Non résolu	366.5
Formulation DFJ avec détection par séparation	8	Non résolu	59	Non résolu	Non résolu	81	98	Non résolu	Non résolu	Non résolu	366.5
Formulation RLT	8.67	49.56	64.56	53.01	64.89	79.94	105.42	152.58	202.25	234.8	346.95
Formulation GCS	8	Non résolu	62.25	Non résolu	Non résolu	94	121.83	Non résolu	Non résolu	Non résolu	205
Formulation SF	**12**	49.48	64.19	64.12	65.29	79.78	105.08	153.83	201.99	236.37	350.77
Formulation MCF	**12**	**71.88**	**75.25**	**79.61**	**101.93**	**107.14**	**151.5**	**321.06**	**403.54**	**464.01**	**433.13**

FIGURE 3 – Valeur objective de la relaxation continue par formulation et instance

Méthodes de résolution	Instance_6_1	Instance_20_1	Instance_20_2	Instance_20_3	Instance_30_1	Instance_40_1	Instance_50_1	Instance_70_1	Instance_80_1	Instance_80_2	Instance_100_1
Formulation MTZ	1	155	72	210	909	109	440	Non résolu	Non résolu	Non résolu	231
Formulation DFJ avec détection naïve	1	Non résolu	102	Non résolu	Non résolu	34	45	Non résolu	Non résolu	Non résolu	7
Formulation DFJ avec détection par séparation	1	Non résolu	102	Non résolu	Non résolu	34	44	Non résolu	Non résolu	Non résolu	7
Formulation RLT	4	163	27	130	88	128	132	242	793	892	228
Formulation GCS	1	Non résolu	43	362	Non résolu	16	31	Non résolu	Non résolu	Non résolu	**1**
Formulation SF	**1**	144	18	112	95	13	142	275	697	948	127
Formulation MCF	1	**31**	**11**	**22**	**15**	**1**	**1**	**110**	**53**	**22**	23

FIGURE 4 – Nombre de noeud du Branch & Cut par formulation et instance

Méthodes de résolution	Instance_6_1	Instance_20_1	Instance_20_2	Instance_20_3	Instance_30_1	Instance_40_1	Instance_50_1	Instance_70_1	Instance_80_1	Instance_80_2	Instance_100_1
Formulation MTZ	0.006	16.205	0.723	6.711	13.706	4.945	59.616	944.198	Non résolu	Non résolu	3.47
Formulation DFJ avec détection naïve	0.007	Non résolu	45.811	Non résolu	Non résolu	19.487	262.524	Non résolu	Non résolu	Non résolu	10.692
Formulation DFJ avec détection par séparation	0.72	Non résolu	42.518	Non résolu	Non résolu	28.636		Non résolu	Non résolu	Non résolu	20.111
Formulation RLT	1.556	2.116	1.361	0.92	0.728	10.48	1.79	39.44	71.97	81.07	**30.85**
Formulation GCS	3.068	Non résolu	24.298	1138.785	Non résolu	9.528	60.85	Non résolu	Non résolu	Non résolu	12.719
Formulation SF	**0.004**	**0.934**	**0.703**	**0.679**	**0.551**	**1.869**	**1.22**	**17.848**	**30.561**	**13.75**	3.306
Formulation MCF	0.007	9.426	3.581	4.588	2.567	5.217	8.252	195.436	169.233	102.069	32.52

FIGURE 5 – Temps de résolution (en seconde) par formulation et instance

3.8 Remarques générales

Nous avons fait le choix de considérer le problème comme non résolu dans le cas où le temps de résolution d'une instance ne prenait plus d'1 heure. Toutes les formulations ont convergées vers une même solution optimale pour chaque instance lorsque celle-ci ont été résolues. Nous remarquons par ailleurs que seules les formulations utilisant des contraintes sur les flows (SF et MCF) ont réussies à résoudre dans un temps raisonnable toutes les instances.

La formulation SF surperforme largement toutes les autres formulations en terme de temps de résolution sur toutes les instances. Ces bonnes performances semblent s'expliquer par le fait que la borne inférieure de la relaxation continue soit plus précise que celles des formulations n'utilisant pas de contraintes de flows et que les itérations du branch & cut soient beaucoup moins coûteuses sur les contraintes de flows que sur les contraintes des formulations polynomiales et exponentielles classiques. La formulation MCF performe également relativement bien grâce à des bornes inférieures de relaxation continue encore plus précises que la SF mais son nombre de variables plus importants détériore ses performances sur les instances avec un grand nombre d'aérodromes. La formulation RLT réalise en outre des performances très intéressantes également grâce à une borne inférieure suiter à la relaxation continue assez proche de la valeur optimal. Par ailleurs, la formulation avec un nombre polynomial de contraintes (MTZ) arrive à résoudre dans des délais raisonnables toutes les instances à part les 80_1 et 80_2. Leurs performances semblent limitées par des bornes inférieures peu précises suite à la relaxation continue. Enfin, les formulations avec un nombre exponentiel de contraintes performent le moins bien, la détection des sous-tours dans la solution temporaire de chaque itération étant un processus très couteux.

3.9 Sensibilité des performances par rapport au nombre d'aérodromes

Nous remarquons quelque soit le choix des formulations que les temps de résolution ne croissent nécessairement pas lorsque le nombre d'aérodromes augmentent. L'instance 100_1 en est un très bon exemple car ses temps de résolotions sont en moyennes beaucoup plus rapides que la plupart des instances de nombre d'aérodromes plus faible. Nous en déduisons que les performances ne sont pas particulièrement sensible au nombre d'aérodrome.

3.10 Sensibilité des performances par rapport aux positions des aérodromes d'arrivée et de départ

Les instances 20_1, 20_2 et 20_3 possèdent les mêmes paramètres exceptés les points de départ et d'arrivée qui diffèrent. Ainsi, nous remarquons que les formulations avec un nombre exponentiel de contraintes présentent des performances très hétérogènes selon les trois instances, échouant notamment à résoudre dans des délais raisonnables les instances 20_1 et 20_3. Ceci peut s'expliquer par le fait que ces instances puissent générer beaucoup de sous-tours selon la choix des points de départ et d'arrivée.

3.11 Sensibilité des performances par rapport à nombre minimal d'aérodrome à parcourir

Les instances 80_1 et 80_2 présentes les mêmes paramètres mise à part le nombre d'aéodromes minimal à parcourir (50 pour 80_1 contre 40 pour 80_2). Nous remarquons à travers les temps de résolution des formulations SF et MCF qu'un nombre minimal d'aéroport à parcourir plus faible semble rendre la résolution plus coûteuse.

3.12 Sensibilité des performances par rapport au nombre de régions

L'instance 70_1 possède un nombre de régions assez élevé par rapport aux autres instances (16 contre 10 au maximum pour les autres instances). Nous remarquons par ailleurs que cette instance est particulièrement couteuse à résoudre quelque soit la formulation. En effet, les formulations MTZ et RLT mettent plus de 15 minutes à la résoudre, les formulations exponentielles n'arrivent pas à la résoudre dans des temps raisonnables et elle constitue l'une des deux instances les plus longues à résoudre pour les deux formulations avec des contraintes de flows. Nous pouvons en déduire qu'un nombre important de régions à visiter peut rendre le problème très couteux à résoudre.

4 Conclusion

Nous pouvons conclure que nous avons modélisé le problème de la Coupe Breitling 100/24 sous la forme d'un programme linéaire en nombres binaires à travers six formulations : 2 formulations avec un nombre polynomial de contraintes (MTZ et RLT), 2 formulations avec un nombre exponentiel de contraintes (DFJ et GCS) et 2 formulations avec des contraintes de flows (SF et MCF). Nous avons ensuite implémenté ces formulations en Julia en utilisant le solveur Gurobi et nous les avons testé sur 11 instances différentes. Les formulations utilisant des contraintes de flows (SF et MCF) se sont montrées les plus performantes en particulier la SF, les formulations avec un nombre exponentiel de contraintes ayant réalisées les plus mauvaises performances en raison d'une détection des sous-tours trop coûteuse. Nous avons également remarqué que la difficulté de résolution d'une instance résidait principalement dans la génération importante de sous-tours qu'elle pouvait engendrer.

5 Annexes

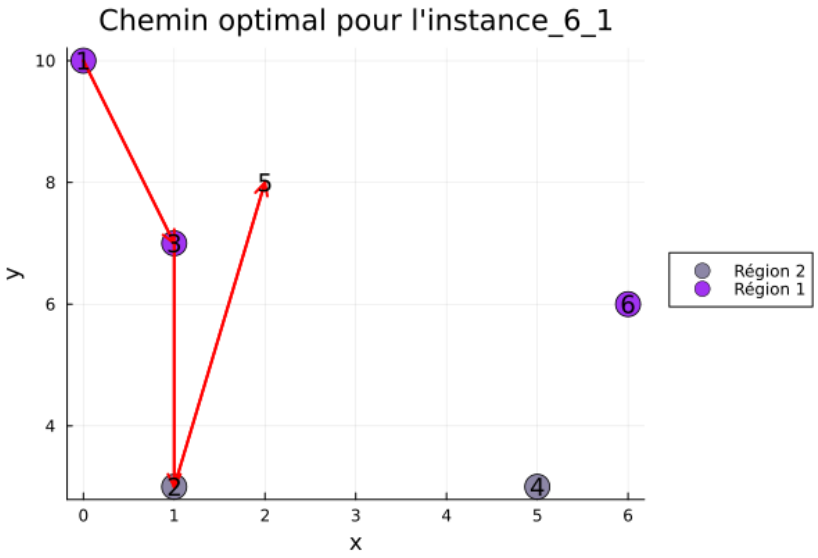

FIGURE 6 – Chemin optimal de l'instance 6_1

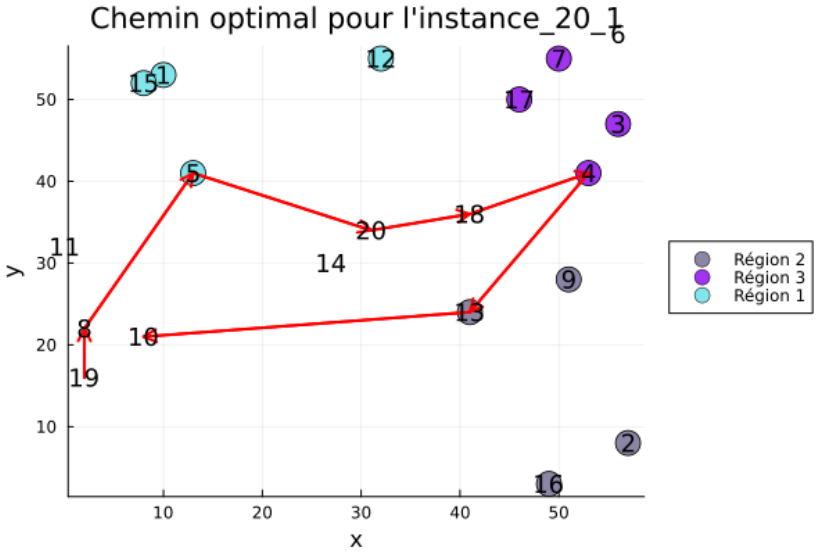

FIGURE 7 – Chemin optimal de l'instance 20_1

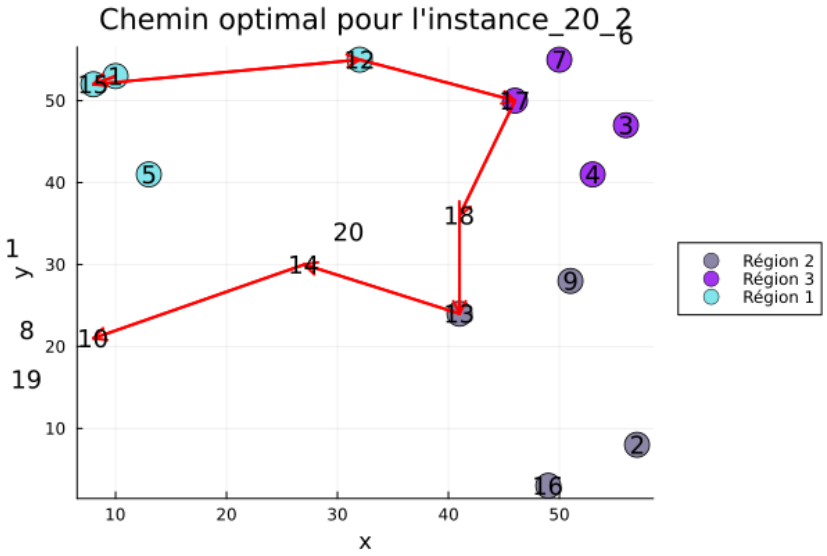

FIGURE 8 – Chemin optimal de l'instance 20_2

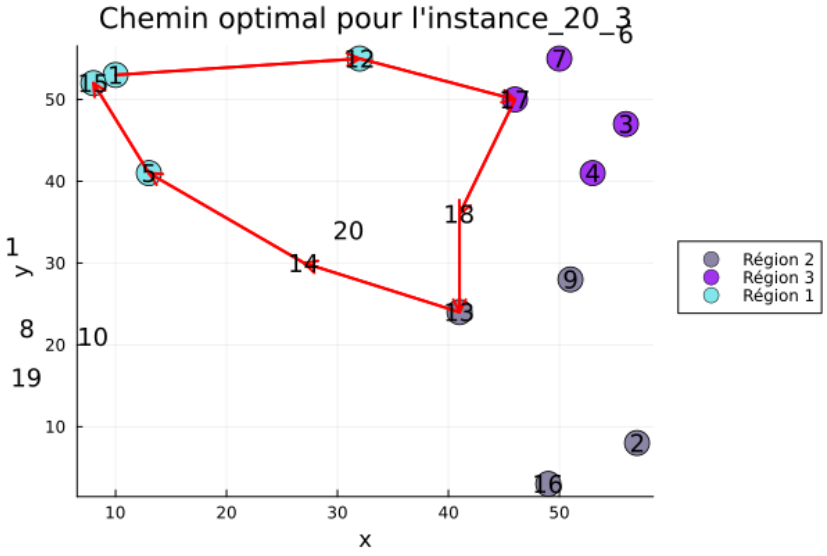

FIGURE 9 – Chemin optimal de l'instance 20_3

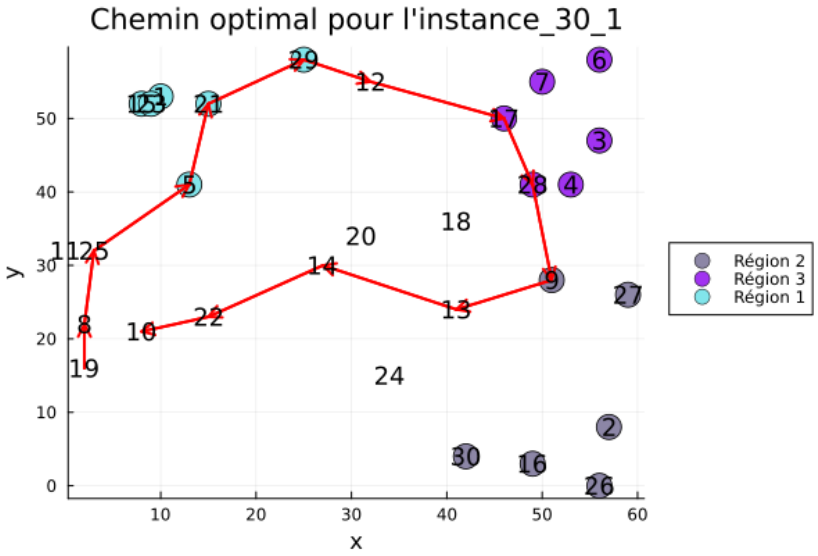

FIGURE 10 – Chemin optimal de l'instance 30_1

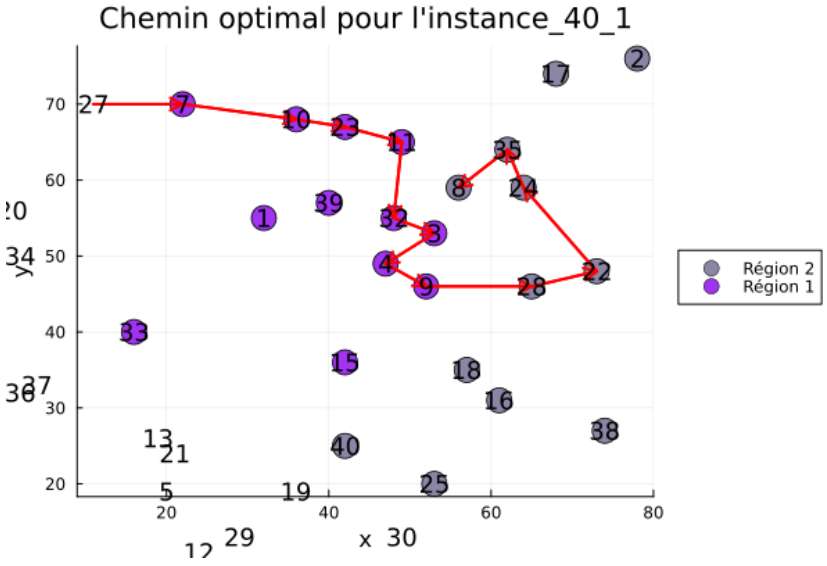

FIGURE 11 – Chemin optimal de l'instance 40_1

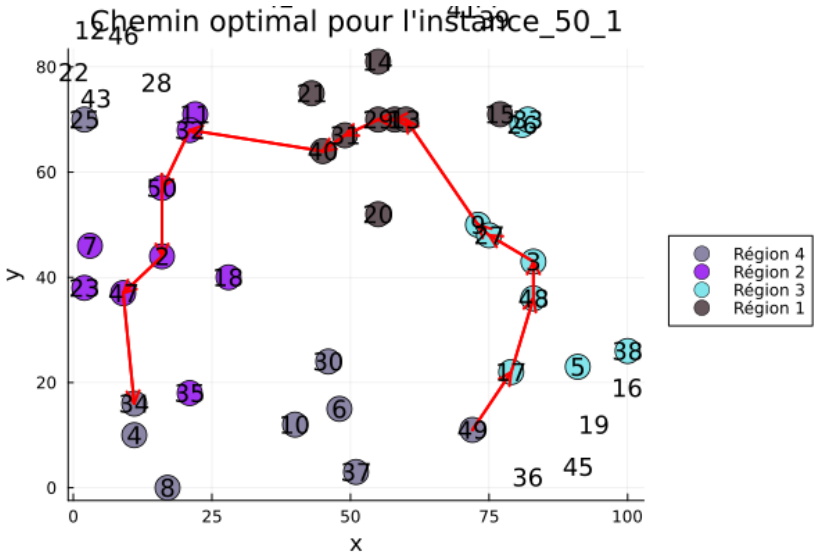

FIGURE 12 – Chemin optimal de l'instance 50_1

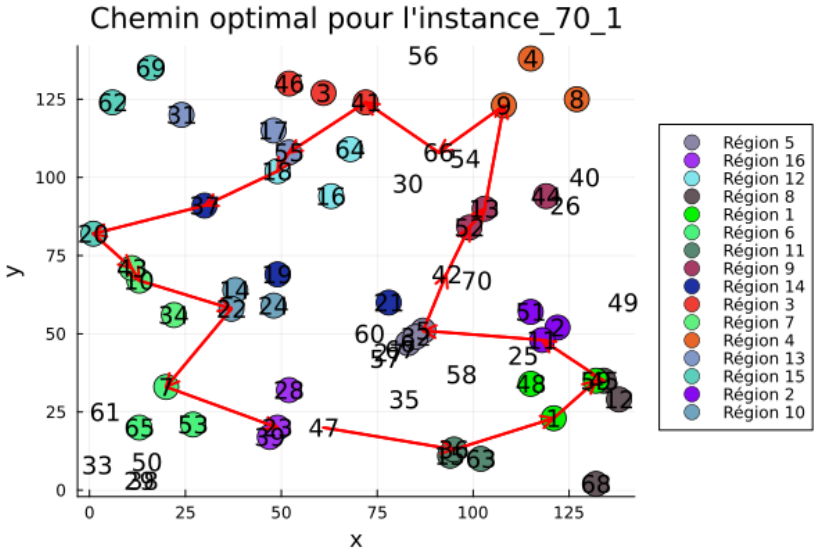

FIGURE 13 – Chemin optimal de l'instance 70_1

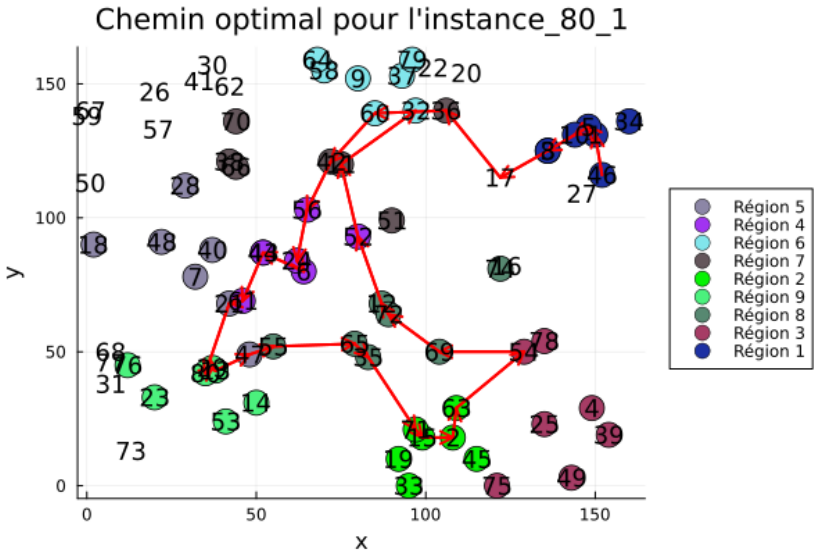

FIGURE 14 – Chemin optimal de l'instance 80_1

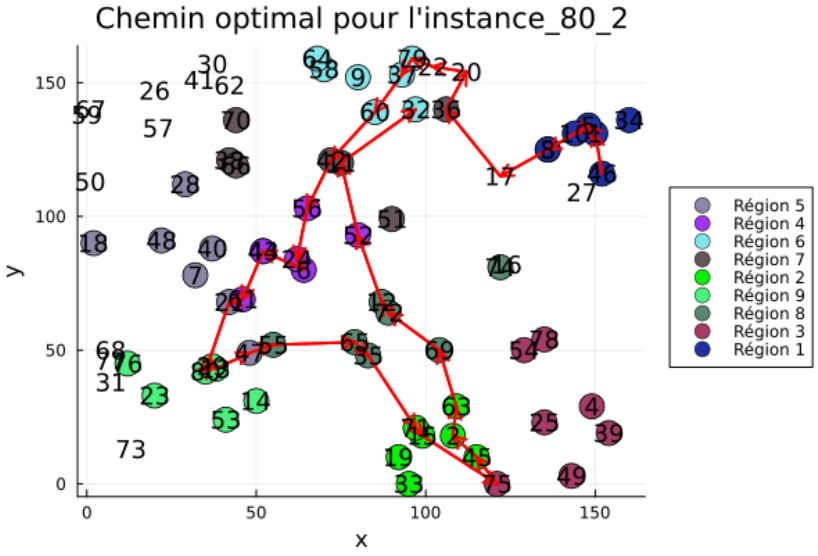

FIGURE 15 – Chemin optimal de l'instance 80_2

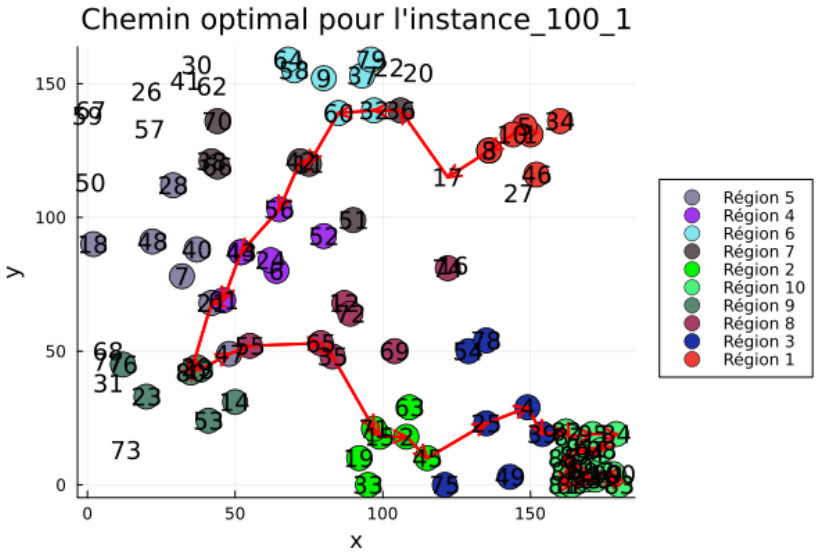

FIGURE 16 – Chemin optimal de l'instance 100_1